AUTOMATIC FACTORIZATION OF GENERALIZED UPPER BOUNDS IN LARGE SCALE OPTIMIZATION PROBLEMS

David Samuel Thomen

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

Automatic Factorization of Generalized Upper Bounds
in Large Scale Optimization Problems

by

David Samuel Thomen

September 1979

Thesis Advisor: Gerald G. Brown

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Automatic Factorization of Generalized Upper Bounds in Large Scale Optimization Problems

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis; September 1979

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

David Samuel Thomen

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
September 1979

13. NUMBER OF PAGES
43

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Generalized upper bounds, GUB, exclusive row structure, ERS, signed identity factorization.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

To solve contemporary large scale linear, integer and mixed integer programming problems, it is often necessary to exploit intrinsic special structure in the model at hand. One commonly used technique is to identify and then to exploit in a basis factorization algorithm a generalized upper bound (GUB) structure (also called a static signed identity basis factorization). This report compares several existing methods for identifying GUB structure. Computer programs have been written to permit comparison of computational efficiency. The GUB programs have

20. continued

been incorporated in an existing optimization system of advanced design and have been tested on a variety of large scale real life optimization problems. The identification of GUB sets of maximum size is shown to be among the class of NP-complete problems; these problems are widely conjectured to be intractable in that no polynominal-time algorithm has been demonstrated for solving them. All the methods discussed in this report are polynominal-time heuristic algorithms that attempt to find, but do not guarantee, GUB sets of maximum size. Bounds for the maximum size of GUB sets are developed, in order to evaluate the effectiveness of the heuristic algorithms.

Approved for public release; distribution unlimited

Automatic Factorization of Generalized Upper Bounds in Large Scale Optimization Problems

by

David Samuel Thomen
Captain, United States Marine Corps
B.A., Alma College, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1979

ABSTRACT

To solve contemporary large scale linear, integer and mixed integer programming problems, it is often necessary to exploit intrinsic special structure in the model at hand. One commonly used technique is to identify and then to exploit in a basis factorization algorithm a generalized upper bound (GUB) structure (also called a static signed identity basis factorization). This report compares several existing methods for identifying GUB structure. Computer programs have been written to permit comparison of computational efficiency. The GUB programs have been incorporated in an existing optimization system of advanced design and have been tested on a variety of large scale real life optimization problems. The identification of GUB sets of maximum size is shown to be among the class of NP-complete problems; these problems are widely conjectured to be intractable in that no polynominal-time algorithm has been demonstrated for solving them. All the methods discussed in this report are polynominal-time heuristic algorithms that attempt to find, but do not guarantee, GUB sets of maximum size. Bounds for the maximum size of GUB sets are developed, in order to evaluate the effectiveness of the heuristic algorithms.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The author wishes to thank Professor Shmuel Zaks for his useful discussions about NP-completeness. He also wishes to thank Captain Bill Wright, U.S.M.C., for his help with the programming effort and insights on the bounding problem.

I. INTRODUCTION

Contemporary mathematical programming models are often so large that direct solution of the associated linear programming (LP) problems with the classical simplex method is prohibitively expensive, if not impossible in a practical sense. It has been found that most of these problems are typically sparse, with relatively few non-zero coefficients, and usually possess very systematic structure. These problems exhibit inherent structural characteristics that can be exploited by specializations of the simplex procedure. Various types of regularity are often described as, for instance, block angular, staircase, and so forth; terms all chosen to describe the visual appearance of the non-zero coefficients when the rows and columns of the problem are conveniently ordered. There are profound economic, managerial and mathematical motives for this special structure, which are not examined here.

Methods to exploit special model structure can be generally categorized as *indirect* (e.g., decomposition), where a solution to the original problem is achieved by dealing with related models which are individually easier to solve, or as *direct*, when the original problem is solved by a modified simplex algorithm.

Among the direct exploitation methods, the most frequently used technique is called *basis factorization*, [6] where the reflection of special problem structure appears and is used to good benefit in the intermediate LP bases. Basis factorization can be *dynamic*, where the algorithm deals with each basis sequentially and/or independently in an attempt to extract as much specialized basis structure as possible, or *static*, where the algorithm depends upon certain types of special structure to be present in *all* bases.

Static basis factorizations include *simple upper bounds*, *generalized upper bounds* (GUB), and *embedded network rows*, among many others. Simple upper bounds are a set of rows for which each row has only one non-zero coefficient. Generalized upper bounds are a set of rows for which each column (restricted to those rows) has at most one non-zero coefficient. Network rows are a set of rows for which each column (restricted to those rows) has at most two non-zero coefficients of opposite sign.

Each of these factorizations permits the simplex algorithm to deal with the static subsets of the rows (and columns) of all bases encountered with prior knowledge that they will satisfy *very* restricted rules. Most of these methods work best when *logic* can be substituted for arithmetic (as is the case with the coefficients ± 1). For this reason, static factorizations often restrict the special structure to possess only ± 1, or to be *scaled* producing an equivalent result.

The concept of Generalized upper bounds was introduced in 1964, the result of work by Dantzig and Van Slyke [4]. The name is derived from analogy to the simple upper bound structure. Graves and McBride [6] refer to *Signed Identity Factorization* as a term more suggestive of the implied basis structure than GUB. Since their introduction, some form of GUB has been implemented in many commercial LP systems. There is often confusion between the mathematical characterization of GUB and these various, widely used implementations of GUB, in that the latter often restrict the GUB set membership rules to permit uncomplicated simplex logic. All of the methods reported here address the full generality of GUB sets but can be modified as necessary to produce restricted GUB sets.

A group of rows collectively form a GUB set (or static identity basis factorization) if they do not *conflict* and can be transformed by simple row and column scaling into GUB constraints. Two rows are said to conflict if there exists at least one variable with non-zero coefficients in both rows.

The details of how GUB structure may be exploited to reduce the computations of the simplex algorithm are not discussed here. See [1, 4, 6, 10,12]. The underlying concept is that the GUB structure enables the simplex algorithm to manipulate the GUB rows implicitly, with logic rather than floating point arithmetic, thus reducing the effective size and solution time for the problem. The more rows one is able to GUB, the fewer rows one has to explicitly carry through the simplex operations. If the original problem has m constraints (of which p are GUB rows) and n variables, then at most only an (m-p x m-p) submatrix of the basis is needed for the explicit simplex operations. This contracted explicit basis means that many of the calculations are replaced with logical operations, yielding faster results and less numerical rounding error. For some problems GUB enables one to construct solutions that might otherwise be beyond capacity of available computers.

A further benefit from GUB is that since at least one variable of every binding GUB constraint must be in the basis, this often suggests an excellent advanced initial basis.

Many problem types have natural GUB structures embedded in them.

a) "Transportation problems (pure, bounded and capacitated networks)

b) Multi-product blending

c) Raw material and/or production resources allocation (forest management, machine loading, plant scheduling)

d) Operations planning (combined production/inventory/distribution planning)

e) Resource assignment (i.e., freight cars, personnel)"[14]

To better illustrate this, Figure (1) contains a presentation of a transportation type problem. Note that a GUB row set has been marked. For problems similar to this, a large GUB set can be quickly found by visual inspection. Likewise, in a particular *class* of models, knowledge of the model structure can lead to problem-independent specification of a proper set of GUB rows. (e.g., one can always GUB all the sinks of a pure network). But for many general models this technique cannot be used, and due to the large size of the problems, visual inspection may be limited to adjacent rows or patterns. This is very dependent on how the problem is written. Most contemporary problems for which GUB factorization may be crucial are so large that an explicit "picture" of the coefficient matrix is not even useful. (One notorious example has been encountered with a "picture" measuring 10 by 300 meters!) It is therefore highly desirable to have an automatic procedure whereby a GUB set can be efficiently identified.

Sources	1	1	1	1	0	0	0	0	
	0	0	0	0	1	1	1	1	
Sinks	−1	0	0	0	0	0	0	0	a set of GUB rows
	0	−1	0	0	−1	0	0	0	
	0	0	−1	0	0	−1	0	0	
	0	0	0	−1	0	0	0	0	
	0	0	0	0	0	0	−1	0	
	0	0	0	0	0	0	0	−1	

Matrix representation of a transportation problem

Figure (1)

In large problems there exist a huge number of subsets of rows that satisfy the GUB criteria. It is generally regarded that those subsets with more rows are "better" GUB sets since they imply a more contracted explicit basis. The implied problem, then, is to find the *maximum* GUB set.

Algorithms to find a maximum GUB row set for a problem do exist. These usually entail enumeration schemes and cannot be guaranteed to be efficient in a practical sense. Conceivably, $2^m - m$ sets of rows might have to be searched before a maximum GUB structure is found. As the problem size grows, the number of possible sets that need to be checked increases *exponentially*. As will be shown later, the hope of finding an efficient algorithm to find the *maximum* GUB set for any general problem is dim.

Therefore, researchers and practitioners have concentrated on constructing efficient *heuristic* algorithms that attempt to identify, but do not guarantee, a maximum GUB set. A few of these methods showing great promise have been reported, but they have not been tested at large scale.

This report outlines several automatic heuristic GUB finding procedures that have been developed and published in the recent literature. These procedures are tested on a suite of large scale, real life optimization problems, and are modified to improve their behavior. Comparative performance of the methods is given both in terms of the computational effort to identify a GUB set, as well as the quality of the GUB set achieved.

Identification of GUB sets of maximum row dimension is shown in Chapter VI to be among the class of NP-complete problems. However, an easily computed *upper bound* on the size of the maximum GUB set is developed and used to objectively evaluate the quality of heuristic GUB algorithms, showing that very nearly maximum GUB sets are routinely achieved.

II. PROBLEM DEFINITION AND REPRESENTATIONS

The Linear Programming problem is defined as

$$(L) \qquad \text{Min } c^t x$$

$$\text{s.t. } \underline{r} \leqslant A x \leqslant \overline{r} \quad \text{(ranged constraints)}$$

$$\underline{b} \leqslant X \leqslant \overline{b} \quad \text{(simple bounds)}$$

where $\underline{r}$ and $\overline{r}$ are m-vectors, x, c, $\underline{b}$ and $\overline{b}$ are n-vectors and A is an m x n matrix. The constraints are sometimes defined as equations, but for the general case of GUB treated here constraints can be equations, inequalities or a mixture. The immediate discussion will be directed at (L); below, the integer and mixed integer problems are treated.

For identification of a GUB set of rows, only the matrix A is used and since the actual values of the non-zero elements of the matrix A are not required the associated matrix K is defined.

$$K \equiv (k_{ij}) \; : \; k_{ij} = \begin{cases} 0 \text{ if } a_{ij} = 0 \\ 1 \text{ otherwise.} \end{cases}$$

As an example, consider the following linear programming problem.

$$\begin{aligned}
\text{Min} \quad & x_1 + 3x_2 - x_3 + 3x_4 + x_5 - 2x_6 \\
\text{s.t.} \quad & x_1 + 2x_2 + x_3 = 4 \\
& 6x_1 + 2x_5 + 5x_6 = 15 \\
& -3x_2 + 2.7x_3 = 7 \\
& 6x_4 + 4x_6 = 8 \\
& 2x_1 + x_5 = 1
\end{aligned}$$

The corresponding matrices A and K are:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 2 & 5 \\ 0 & -3 & 2.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 4 \\ 2 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$K = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

There are several ways one can model the maximum GUB problem. Three approaches are presented to aid in the understanding of the theoretical context of the heuristic methods examined and to highlight the formal complexity of the original problem.

Two rows of K (or A) are said to *conflict* if there exists at least one column with non-zero coefficients in both rows. The GUB problem can be restated as that of finding a subset of the rows that do not conflict.

A. GRAPH THEORY REPRESENTATION

Consider the matrix K of the linear programming problem (L). A graphical representation of this matrix can be constructed through the following mapping rule, f. Let each row of K be a vertex of the graph. Should two rows of K conflict then the two vertices of the graph are joined by an edge. This mapping retains all the necessary conflict information.

The graph associated with the example is presented in Figure (2). Note that it has five vertices, one for each row. Since row 1 conflicts with rows 2, 3, and 5, edges connect vertex 1 to those vertices. The other edges represent the remaining conflicts. If two vertices, a and b, are joined by an edge, e, then a and b are adjacent, and a (or b) is *incident with* e. Since vertices 2 and 3 are not adjacent, this indicates that the corresponding two rows in K do not conflict.

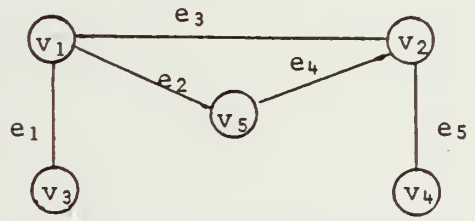

Figure 2

This introduces the notion of *independence.* Given a graph G = (V, E), a subset V' V is said to be an *independent set* if no two of its elements are adjacent. It follows that if an independent set of vertices can be found in G then the corresponding rows of the matrix K do not conflict and thus define a GUB set. Conversely, a GUB set for K defines an independent set for

the graph G. It is also clear that an independent set for G is maximum if and only if the corresponding GUB set for K is maximum.

In the example problem, the maximum independent set is 3, 4, 5 . These are also the rows of the maximum GUB structure.

Consider the set K_m, the set of all K-type matrices having m rows. The above mapping factors this set into a finite number of *classes*. Two matrices, K_1 and K_2 are said to belong to the same class, C, if and only if each is mapped into the same graph, G_c.

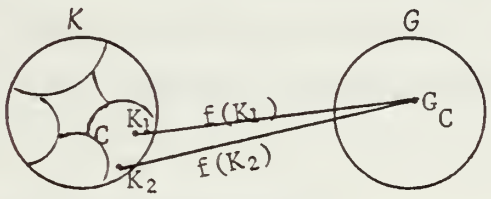

Figure 3

Thus, an independent set of vertices of G_c correspond to a GUB row set for every matrix in the class C.

The incidence matrix N is defined as follows.

$$N \equiv (n_{ij}) : n_{ij} = \begin{cases} 1 \text{ if vertex i is incident with edge j} \\ 0 \text{ otherwise} \end{cases}$$

For the example problem N would be:

$$N = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

There exists one, and only one incidence matrix for each graph of G, where G, is the set of all graphs having m vertices.

Since the set of all N-type matrices with m rows is a subset of K_m, every class of K_m contains one and only one incidence matrix. In general, for the GUB problem, every m row matrix is equivalant to one of a finite number of incidence matrices. Superficially this may seem to be a simplification. But as shown in Chapter VI the GUB problem on N is as difficult as the independent set problem on G. The equivalent statements of the GUB problem do, however, offer different views of the problem which are helpful in considering algorithms for and analysis of the problem. [Note: In Garey and Johnson [5] it is shown that two other graph problems, the "vertex cover" and the "clique" problem, are equivalent to the independence problem, and hence the GUB problem. These problems do not seem to offer any additional insight for the GUB problem.]

B. CONFLICT MATRIX REPRESENTATION.

The first method is developed around a *conflict* matrix. This is a square matrix of dimension m, defined by:

$$M \equiv (m_{ij}) \; : \; m_{ij} = \begin{cases} 1 \text{ if row i conflicts with row j in (L)} \\ 0 \text{ otherwise.} \end{cases}$$

For the example problem:

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Note that this matrix is symmetric. The sum for any row (or column) indicates the number of other rows it is in conflict with, plus one.

This sum is important in that it indicates for any particular row how many other rows would be subsequently excluded from a GUB set by its addition.

The rows of a GUB structure can be rearranged to form an embedded identity matrix in M. Note that this is the case for rows 3, 4, and 5 in the example problem.

C. VECTOR SPACE REPRESENTATION

The second heuristic approach can be modeled using vectors in an n-dimensional vector space, where n is the number of variables in the problem (L). Consider each row of K as a vector in this space, having unit length in those "dimensions" corresponding with its non-zero coefficients. In the example problem, row 1 is represented by the following vector:

$$(1, 1, 1, 0, 0, 0,)$$

R, the resultant vector from the sum of all the vectors of the rows of K, indicates the number of conflicts, plus one, associated with each variable of (L). A hyper cube in n-space situated in the first orthant at the origin with length 1 in all positive directions denotes the feasible GUB region. Should R extend beyond this area, then the set of rows corresponding to the vectors determining R does not constitute a GUB structure.

A gradient vector can be calculated indicating the direction of the shortest distance to the *feasible region*. It can be used to determine which row to remove from the set to obtain the largest movement on the desired direction. When R falls within the feasible region, the set of rows determining R constitutes a GUB set.

III. EARLIER LITERATURE

Two papers dealing with efficient GUB finding methods are worthy of special note.

Brearly, Mitra and Williams [2] establish a very useful framework for study of methods for finding GUB structure, as well as an insightful discussion of these methods and a taxonomy for their classification.

They define three sets consisting of the rows of the technological matrix A. The first set, the *eligible set*, is made up of every row of A that is individually eligible to belong in the GUB set. The *structure set* is a subset of the eligible set and includes all those rows currently considered as members of the GUB set. The *candidate set* consists of those rows of the eligible set that are candidates for inclusion (or re-inclusion) in the GUB set. Every one of the methods examined in [2] is described in terms of manipulation of these sets.

Each method of building a GUB set employs one of two basic strategies. The *type I* (row addition) strategy assumes initially that none of the rows belong to the GUB set. Then, based on a particular type I criteria for inclusion, rows are removed from the candidate set and either added to the structure set or dropped from further consideration. This procedure continues until the candidate set is empty. The rows in the structure set form an admissable GUB structure.

The *type II* (row deletion) strategy takes the opposite approach and is divided into two phases. Methods of this type assume initially that all the eligible rows are elements of the structure set. This assumption normally leads to an infeasible GUB set with many conflicting rows. Based upon the particular type II decision rules, rows are removed from the structure set and placed in the candidate set. The first phase of this strategy ends when a feasible structure is obtained.

The second phase involves examining the removed rows in the candidate set. Those that do not conflict with any of the members of the current structure set are taken from the candidate set and re-included in the structure set. Those that do conflict are deleted from the candidate set and dropped from further consideration. The second phase ends when the candidate set is empty. At this point the rows of the structure set consistute an admissable GUB set.

Brearly, Mitra, and Williams examine over 18 different methods. These approaches differ in the primary and secondary decision criteria for including (or removing) a row in the GUB structure set. The heuristic decision rules examined are based on the following model entities and combinations thereof:

Include or remove a row based upon:

a) the number of non-zero elements in the given row,

b) the number of rows in conflict with the given row,

c) the number of non-zero elements in rows that conflict with the given row,

d) the row's relative weight obtained by the inner product of a vector representation of the row and a directional gradient.

Except for the last, these rules were tested with both strategies. As an example, I.1 (type I strategy, method number 1) has as its primary decision rule for adding rows to the structure set: choose a row from the candidate set with the minimum number of non-zero elements. Method II.1 removes rows from the structure set based upon the maximum non-zero element count for each row.

These methods were implemented with an ALGOL program run on an ICL 4130 computer. Several linear programming models were run with this program and the results (of 12) were presented. These problems range in size from 12 rows up to 166 rows.

The results show that those methods using heuristic (d) above "consistently performed very well" [2]. Similarly, those methods using heuristic (b) were found to perform nearly as well as (d).

McBride [15] compares the directional gradient method (d) with an approach suggested, but not tested by Greenberg and Rarick [7]. The latter method uses the conflict matrix as does heuristic (b). However, it focuses on finding a maximum embedded identity matrix within the conflict matrix, rather than using the conflict matrix to determine conflict counts, applying a specialization of the preassigned pivot procedure (P^3) normally used for reinversion [8]. McBride's results indicate that heuristic (d) is significantly faster. However, neither method consistently achieves a larger GUB set.

McBride also comments on the notion of a "good" GUB set. He finds merit in selecting a set of GUB rows that minimizes the non-zero build-up in the representation of the inverse transformation of the explicit basis, during actual optimization. Results are also given for a restricted GUB set selection that gives priority to equality constraints. Since equality constraints are always binding in feasible solutions, the subset of the basis associated with binding constraints, or kernel [6] is expected to have less explicit non-zero elements.

Based upon the results in these papers, and on independent computational experience with automatic GUB factorization reported by Brown and Graves [3], the present research was initially concentrated on those approaches utilizing the two most successful heuristics (I.2, II.10 and variations).

The models studied in this report are of a larger scale and include mixed integer problems as well as models for which prior GUB row sets have been manually specified.

Most of the notation and labels of [2] have been retained here for their clarity.

IV. DETERMINATION OF THE ELIGIBLE SET

The implementation of GUB in simplex algorithms usually admits only ± 1 as non-zero coefficients in the GUB rows. In linear programming a scaling of columns can make each non-zero element in a GUB row ± 1. For variables of an integer or mixed integer programming problem, the columns of matrix A that correspond to integer variables can not be scaled without destroying the integrality condition. Therefore, non-zero elements in columns corresponding to integer variables can be modified only by row scaling. If it is impossible to obtain the necessary ± 1 non-zero coefficients by row scaling and column scaling of columns corresponding to continuous valued variables, the row is not eligible for inclusion in a GUB set.

To provide the complete context of this research, the procedures examined for locating a GUB set in a linear programming problem are designed to be incorporated as an automatic, integral part of a contemporary optimizing system of advanced design.

Each method is implemented as a feature of the read routine (written to accept input in the standard MPS format, as well as editing information indicating integer variables, scaling

and known prior GUB structure). Each method automatically examines the rows of the input and specifies a GUB set. The appropriate rows and columns are then scaled as necessary to obtain the proper GUB structure, and passed on to the optimizing portion of the system. (Note that the editing information places conditions that must be satisfied for any achievable GUB set.)

In determining the set of eligible rows, the following factors have to be considered.

a. Through the editing process, have some of the rows been dropped from the problem? If so, these rows are not eligible for inclusion in the GUB structure and are thus dropped from the set of eligible rows.

b. Through the editing process, have any rows been predesignated to be in the GUB structure? (As previously mentioned, large segments of the constraints can often be selected for the GUB set either visually or by the implicit nature of the type of problem.) Any rows that conflict with these rows are not eligible for subsequent inclusion.

c. All those rows that are designated "nonconstrained" (N) (which include the objective function) are not eligible for inclusion in the GUB structure. All such rows, other than the objective function, are subsequently handled independently of the optimization.

d. If there are any integer valued variables, an additional check is performed. A row in the GUB set must eventually be capable of being scaled to ± 1 non-zero coefficients. This is achieved, if necessary, through a combination of row and column scaling. However, with integer variables, column scaling is no longer advisable. Therefore any row with a non-zero element in integer columns that is not a +1 or −1, or capable of being rendered into a ± 1 in those positions through row scaling alone, must be marked as ineligible for inclusion in the GUB structure. Figure (4) gives the flow chart of how this procedure is implemented.

Once the above restrictions have been considered, the resulting set of eligible rows is then available for search in order to construct the desired GUB structure.

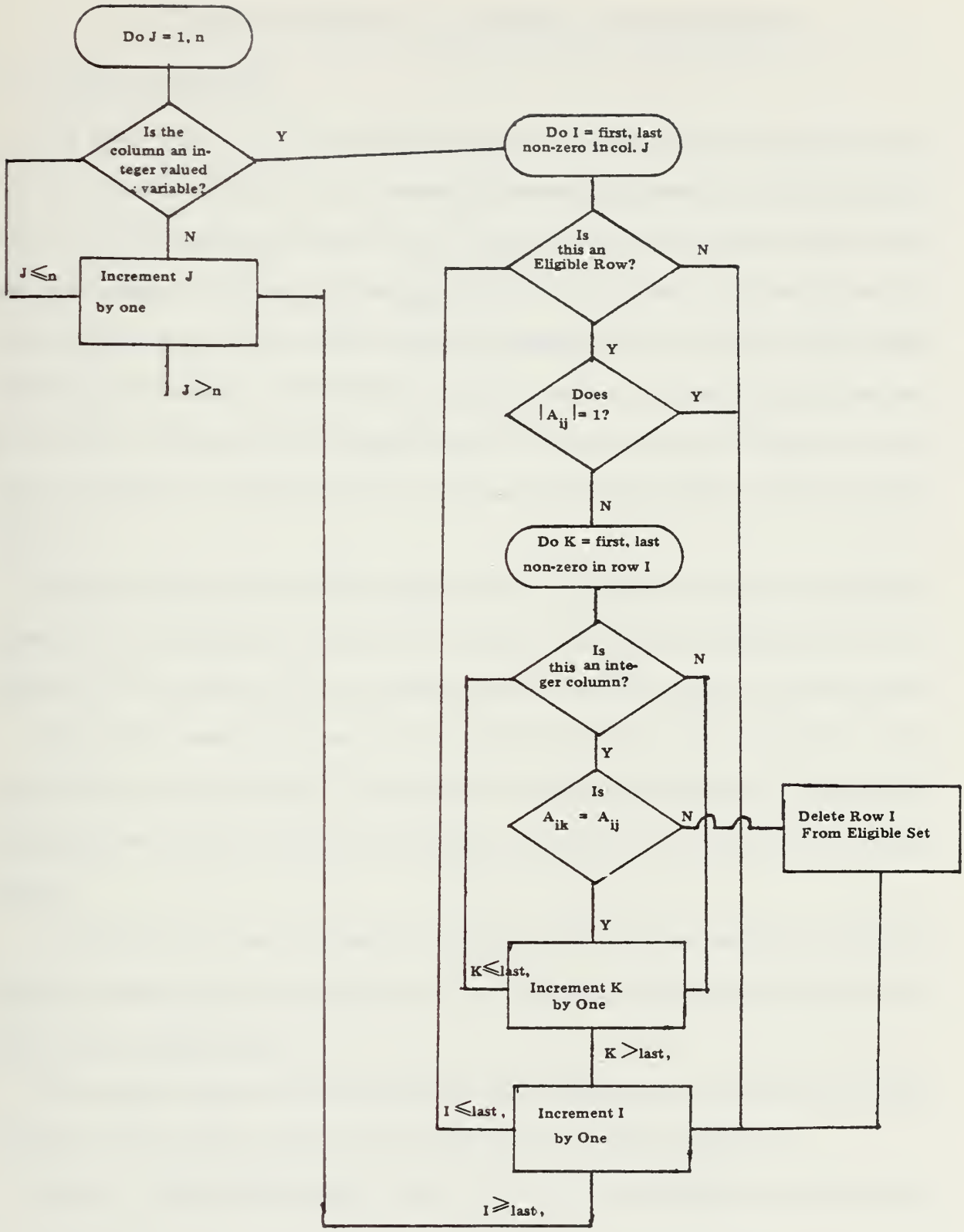
Do J = 1, n
Is the column an integer valued variable?
Y
N
Do I = first, last non-zero in col. J
Is this an Eligible Row?
N
Y
Does $|A_{ij}| = 1$?
Y
N
Do K = first, last non-zero in row I
Is this an integer column?
N
Y
Is $A_{ik} = A_{ij}$
N
Delete Row I From Eligible Set
Y
K ⩽ last, Increment K by One
K > last,
I ⩽ last, Increment I by One
I ⩾ last,
J ⩽ n Increment J by one
J > n

Figure 4

V. IMPLEMENTATION OF AUTOMATIC GUB HEURISTICS

A. CONFLICT METHODS

The approaches I.2 and II.2 employ the notion of a conflict measure for each row. Consider the conflict matrix, M, of the corresponding technological matrix A, for which a GUB set is to be found. An individual element, m_{ik} is 1 if row i and row k of the original matrix have at least one column j such that $a_{ij} \neq 0$ and $a_{kj} \neq 0$. If the two rows share no non-zero column then the corresponding m_{ik} of the conflict matrix is 0. Summing across a row of the conflict matrix can thus give the measure of the number of rows plus one that are in conflict with a given row. For a given row, this sum less one indicates exactly how many other rows would be excluded from the GUB set by inclusion of this row. This second number is called the row's *deletion potential.*

Method I.2 initially places all the eligible rows on a candidate list. From the candidate list, individual rows are selected and removed to be added to the structure. Other rows that are in conflict with the selected row are immediately removed from the candidate list and discarded.

The heuristic selects those rows on the candidate list with the *minimum* deletion potential to be added to the structure first. The selection of rows for the structure and the discarding of conflicting rows continues until the candidate list is exhausted. The resulting structure forms a GUB set.

A modification to the above heuristic is possible which *breaks ties* among rows sharing the minimum deletion potential by selecting the row having the most non-zero elements for inclusion into the GUB structure.

The program used to test this heuristic approach is adapted from an earlier version made available by Glenn Graves. A step by step description of the method is given below.

Step 1. Identify Eligible Rows. Set $\beta_i = 1$ if row i is an eligible row, and equal to 0 otherwise.

Step 2. Determine Deletion Potential. Scan each eligible row i and increment β_i by the number of other eligible rows k, where a_{ij} and a_{kj} are both non-zero for at least one column j. (β_i is the deletion potential, plus one.)

Step 3. Stopping Condition. If all the $\beta_i = 0$, stop. Otherwise, go to the next step.

Step 4. Row Selection. Select row i having the minimum positive ("deletion potential") β_i and add it to the structure.

Step 5. Exclude Rows in Conflict with Selected Row. Locate the $(\beta_i - 1)$ rows in conflict with the selected row i. For each of these rows k, locate the $(\beta_k - 1)$ rows that they are in conflict with and decrement β_i for those rows by one.

Step 6. Marking Selected and Excluded Rows Ineligible for Further Consideration. Set β_i and the β_k's equal to zero. Go to step number 3.

Only rows with $\beta_i > 0$ are eligible. In step 1 β_i is set to 1 for eligible rows. In the next step the β's for these rows are modified by each row's deletion potential. Assuming there are still some eligible rows, the one with the smallest deletion potential is selected in step 4 for inclusion in the structure. In the next step all the rows conflicting with the one selected are identified for discard and the deletion potentials of the remaining rows are updated. In the last step, both the selected row's weight and those of the discarded rows are set equal to zero. When all rows have either been selected or discarded the β array will be all 0's. At this point the selected rows form a GUB structure.

Method II.2 (row deletion) initially places all the eligible rows in the structure set. From this set individual rows are selected and placed on the candidate list in order of *maximum* deletion potential. During Phase 2 Brearly, Mitra, and Williams drop all rows from further consideration that conflict with the structure set and attempt to re-include remaining candidate rows (that do not conflict with the structure set) in LOFI order. A modification of phase 2 is used in this research which simply excludes from further consideration all conflicting rows, re-includes any remaining candidate rows, and repeats phase 1, until no further non-conflicting candidates remain.

B. GRADIENT METHODS

The second method (II.10) employs a heuristic method put forth by Senju and Toyoda [16] for approximate solution of certain linear programming problems with 0, 1 variables. The general problem that they address is that of choosing a most profitable combination (or portfolio) of orders subject to resource constraints and an all-or-nothing (0-1) restriction on the orders. (i.e., an order is not allowed to be only partially filled.)

This same format can be used to express the search for a maximal GUB structure in (L). The rows of the technological matrix A are treated like the orders in the Senju and Toyoda model in that they are to be either included with or excluded from the GUB set. The objective is to obtain a maximum number of rows in the GUB structure while satisfying the stipulation that the GUB rows be disjoint. This last restriction can be expressed as a set of resource restrictions in the sense of Senju and Toyoda.

In mathematical terms, the GUB finding problem can be formulated as follows:

$$(S) \quad \text{Max } Z = x_1 + x_2 + \ldots + x_m$$

subject to

$$\sum_i k_{ij} x_i \leqslant 1 \qquad j = 1, \ldots, n$$

$$x_i = 0 \text{ or } 1 \text{ for all } i$$

where:

m: is the number of candidate rows in (L),

n: is the number of variables in (L),

k_{ij}: is the (i,j) element of the matrix K, which in turn is the 0, 1 matrix associated with the matrix A,

x_i: is the variable which determines if row i is in the GUB set or not,

Z: is the objective function.

Senju and Toyoda outline a heuristic approach for obtaining a near-optimal solution for the problem they examine. Adapting their approach to the specialization (S) given above, a type II strategy results, with all the rows initially being included in the GUB structure. Using the constructive characterization of a vector space outlined earlier, consider each row of (S) as a vector in n-space. [n is the number of variables in (L)]. A resultant vector R is determined by the sum of all the included rows and, in general, extends beyond the feasible space denoted by the unit hyper cube. A gradient vector is calculated from this infeasible point in the direction of the shortest distance to the feasible region. In Brearly, Mitra and Williams [2] this vector is labeled σ^{II}. An inner product of this gradient with each of the row vectors results in a relative weight for each row. These weights, which are stored in a vector labeled ν^{II} can be viewed as indicating the relative contribution that the removal of the corresponding row would have towards obtaining a feasible structure.

Rows are removed from the structure set according to their relative weight, the largest weight being removed first. This process is continued until a feasible set of GUB rows has been obtained. (The gradient vector is not recomputed as the method proceeds.)

Next, a phase 2 procedure is implemented which examines each of the initially removed rows to see if any can be re-included into the structure without violating the bounds of the unit hyper cube. Upon completion of phase 2, the selected rows constitute a GUB set.

A variation on the above procedure recalculates the shortest distance to the feasible region after the removal of each row. With the new gradient, a new set of relative weights for the remaining rows is then calculated and used, if necessary, to determine which of the subsequent rows will be removed. This method is named II.9.

Another modification is possible if two rows are found with equal weights. As a tie-breaking rule, the row found to have the least number of non-zero coefficients is discarded first.

A step by step outline of the heuristic approach follows:

Phase I : Deletion of Infeasible Rows

Step 0. Initialize Sets. Add all eligible rows to the structure set. The candidate set is empty.

Step 1. Determining the Vector R. For each column j, define ρ_j as the number of rows in the structure set having non-zero elements in column j.

Step 2. Determining Relative Weight of each Row. For each row i, define ν_i as the sum of the $(\rho_j - 1)$ of every column j, for which $a_{ij} \neq 0$.

Step 3. Feasibility Condition. If, for every column, $\rho_j \leqslant 1$, then go to step 6; else find a column j such that $\rho_j > 1$.

Step 4. Determining Row for Exclusion. Examine the rows in the structure having non-zero elements in column j. Select the row i with the largest ν_i.

Step 5. Removal of Selected Row. Remove row i from the structure set, decrementing ρ_j by one for every column j with $a_{ij} \neq 0$. Add row i to the candidate set and return to step 3.

Phase 2 : Improving on Feasible GUB set Found by Re-including Excluded Rows

Step 6. Eliminate Rows in Candidate Set that Conflict with the Feasible Set. For every row i of the candidate set that has at least one $a_{ij} \neq 0$ in a column with $\rho_j = 1$, remove that row from the candidate set.

Step 7. Re-inclusion of Row. If any rows remain in the candidate set, then find row i having the smallest ν_i. Remove row i from the candidate set and re-include it in the structure set. Increment ρ_j by one for every column j where $a_{ij} \neq 0$.

Step 8. Stopping Condition. If the candidate set is empty, stop; else go to step 6.

In step one, the vector ρ is calculated as the sum of all the individual row vectors of m. Step two calculates the relative weights that result from the inner product of the gradient vector with each of the row vectors. These are stored in the array ν . The next step examines ρ to see if the vector is within the feasible region. If not, a row with the largest relative weight is removed from the structure set and the ρ vector is updated to reflect the sum of the row vectors remaining in the structure set.

Once a feasible structure has been obtained, the candidate set (which consists of those rows initially removed) is scanned in step 6. Any of those rows found to still be in conflict with the rows of the structure set are discarded. Among those rows which remain, that with the smallest relative weight is re-included in the structure. This cycle of discarding and re-inclusion is continued until the candidate set has been emptied. The resulting rows of the structure set constitute a feasible GUB set.

To modify the algorithm in order to compute a new gradient vector after the removal of each row in phase 1, step 5 is changed as follows:

Step 5.* Removal of Selected Row. Remove row i from the structure, decrementing ρ_j by one for every column j such that $a_{ij} \neq 0$. Locate each row k that is in conflict with row i. Decrement ν_k by the number of conflicts between the two rows. Add row i to the candidate set and return to step 3.

Now when a row is removed from the structure set, the ν_i contain the new relative weights equal to the inner product between the vector for row i and the new gradient.

These two basic methods have been implemented as integral modules of large scale optimization system. Therefore, explicit conflict matrices are not built. (To have done so would have consumed too much computer time and space.) Instead, all the information is stored in the vectors β, ρ, and ν.

Logical flags associated with each row indicate whether it is eligible, and whether it is in the candidate set or in the structure set.

As mentioned previously, the problem data is read in MPS format and expressed internally in terms of only the non-zero elements. This input is stored in a doubly linked list having both a row and a column thread. Thus, along with any non-zero coefficient a_{ij}, the location of adjacent non-zero elements in both the row i and column j are also immediately available. This crucial feature permits efficient row access for various operations (e.g., to locate all rows that conflict with a given row at a particular column.)

C. COMPUTATIONAL RESULTS

The heuristic methods have been tested on fifteen real-life problems that vary in size from 92 constraints to 4, 648 constraints. A description of each of the problems in given in figure (5). As can be seen, four of the problems are mixed integer and two are pure integer. The experiments have been conducted using the FORTRAN H compiler on an IBM 360/67 computer at the W.R. Church computer center of the Naval Postgraduate School. All execution times reported are expressed in actual CPU seconds, accurate to the precision displayed.

The results of these experiments are given in Appendix A. The first two columns give the rows and non-zero column elements, respectively, of the GUB structures found. The time given in column three is the time required to locate the GUB set once the set of eligible rows has been determined. The final columns give additional information relating to the two versions of the gradient methods examined and represents total time in phase 1 and the number of rows reincluded in the GUB structure during the phase 2.

Problem	Number of rows	Number of columns	Integer Columns	Non-Zeros
VANN	92	1,324	1,324	2,648
NETTING	103	247	103	494
AIRLP	171	3,040	0	6,023
COAL	171	3,753	0	7,506
TRUCK	239	4,752	4,752	30,074
CUPS	415	619	145	1,341
FERT	606	9,024	0	40,484
PIES	663	2,923	0	13,288
PAD	695	2,934	0	13,459
ELEC	785	2,800	0	8,462
GAS	799	5,536	0	27,474
FOAM	1,017	4,020	42	17,187
LANG	1,236	1,425	0	22,028
JCAP	2,487	3,849	560	9,510
ODSAS	4,648	4,683	0	30,520

Figure 5

As with the earlier work cited, the Senju and Toyoda method has been found to be consistently the fastest. In general this holds true for both method II.9 and II.10. II.9, which updates the gradient after each row is removed, takes longer in phase 1 than its counterpart. However, it so selectively deletes the rows, that few if any rows are ever added back into the structure during phase 2. This suggests the possibility of implementing II.9 as only a one phase method.

All methods are robust in that they find large GUB sets. The conflict approaches generally find a larger number of variables with non-zero coefficients in the GUB rows. However, this approach definitely becomes inefficient when larger problems are analyzed, regardless of the relative size of the GUB structure in the problem.

There is some discrepancy between these results and those published earlier [2], especially with regard to the times of the other methods compared to II.10. The wide discrepancy between II.9 and II.10 has not been observed in the current research. It is hypothesized that this is due partially to differences in implementation of the various approaches and partially to problem size and structure variations between these studies.

VI. PROBLEM COMPLEXITY

The *complexity* of a problem is said to be polynomial if an algorithm exists for which the fundamental operations are limited by a polynomial function of instrinsic problem dimensions. Such an algorithm would be called a *polynomial time* or *good* algorithm. The class of all problems for which such algorithms exist is denoted (P). If an algorithm is not polynomial time, then it is defined to be an *exponential time* algorithm. The disadvantage of an exponential algorithm is seen in the explosive growth of the maximum solution time relative to a good algorithm as the dimensions of the problem increase [13].

A problem x is said to be *reducible* to a problem y if each *good* algorithm for solving y can be used to produce in polynomial time a good algorithm for solving x [11]. Note that this does not necessarily require that a good algorithm for x and y actually exist. This requires only that if one exists for y, then one also exists for x.

An *intractable* problem is one for which it is known that no polynomial time algorithm exists. In between this class of problem, and the class P, is a vast number of problems whose status is uncertain. Among these is a class of *nondeterministic polynomial-time* problems (NP) for which a polynomial time algorithm can be shown to exist that can *verify* a guessed solution, but for which the existence of a (determininstic) polynomial time algorithm to actually solve a problem has not yet been demonstrated.

If every problem of the class NP is reducible to the problem y, then y is said to be *NP-hard.* In addition, if y itself belongs to NP, then y is *NP-complete* [5, 11].

The following problem is known on the literature as the independent set decision problem (ISD). It belongs to the set of NP-complete problems.

(ISD) Given a graph G = (V,E) and an integer t, does G contain an independent set of size t or more. The GUB decision problem GUBD can be defined as follows:

(GUBD) Given an m x n 0,1 matrix K and an integer p, does K contain a set of p or more rows $i_1, i_2 \ldots, i_q$ such that

$$(*) \sum_{e=1}^{q} k_{i_e j} \leqslant 1 \quad \text{for every column } j; \quad q \geqslant p.$$

Given an instance of the ISD problem, the incidence matrix N can be constructed. This matrix along with the integer t is an instance of the GUBD problem. The following theorem proves the correctness of this reduction:

Theorem: The incidence matrix N has t rows satisfying (*) if and only if there are t vertices in G that are independent.

Proof: a) Assume there exists t rows of N that satisfy (*). They correspond to vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_t}$ in G. If any two of these vertices are adjacent, then

$$\sum_{e=1}^{t} n_{i_e j} = 2$$

where j is the column in N that corresponds to the edge connecting the two vertices. This is a violation of the assumption, hence the t vertices in G are not connected to one another.

b) Assume there exists t vertices $v_{i_1}, v_{i_2}, \ldots v_{i_t}$ in G that are independent. Since no two are adjacent, the corresponding rows in N satisfy (*). Q.E.D. [18]

Since the ISD problem, a problem known to be NP-complete, is reducible to the GUBD problem, it follows that the GUBD problem itself is NP-complete. (It is clear that the reduction is polynomial time and it is also clear that GUBD is in NP.)

The related problems of finding a maximum independent set and a maximum GUB set are not in NP, however, they are NP-hard. It is therefore unlikely that a polynomial-time algorithm will be found for these problems. Only exponential-time algorithms are presently available. The above analysis of GUB algorithms has only indicated the *worst case* bound. No conclusions are made about the *expected* (i.e., average) performance of an algorithm. In other words, the possibility of the existence of an algorithm with a good expected performance times, but having an exponential worst case bound, has not been ruled out.

VII. AN UPPER BOUND FOR THE SIZE OF MAXIMUM GUB SET

The intrinsic difficulty of identifying a maximum GUB row set has been shown to be exponential, making this task essentially impossible for problems of the scale at hand. However, the efficient heuristic procedures have been shown to provide very large GUB sets, whose size appears to be relatively stable for each problem regardless of the particular method applied. This suggests that these large GUB sets may be, in fact, very nearly maximum, although there is no practical way to verify this directly.

Although the problem of determining the size of the maximum GUB set is also NP-hard, it is possible to develop an easily computable *upper bound* on the maximum GUB set size. This bound can then be used to objectively evaluate the quality of the GUB sets produced by heuristic algorithms.

It is clear that the number of rows of a GUB set can be no greater than the number of rows in the problem. Also any one row by itself can form a GUB set. But these bounds are of little practical use when considering the problem of identifying a maximum GUB set. Utilizing information that is already available in the heuristic procedure, it is possible to construct in polynomial time an upper bound on the size of the maximum GUB set. (It is also possible to construct a lower bound on the size of the maximum GUB set, but that topic is not pursued in this report.)

For the purpose of developing a better bound, the incidence matrix representation (N) of the problem is used. Let s_i be the number of 1's in row i. Note that s_i is the number of edges incident to vertex i in G. Also note that $s_i = \beta_i - 1$. The number of columns in N represents the number of distinct conflicts that exist between the rows of the original problem. This number is denoted as c, and can be found by the following formula.

$$c = \frac{\sum_{i=1}^{m} s_i}{2} .$$

If c is greater than 0, all the rows of N cannot simultaneously belong to a GUB set, which implies the cardinality of the GUB set is less than m. As c becomes larger, the following argument shows that the upper bound of the maximum GUB set decreases.

If c is positive, but strictly less than m, it is possible for all the conflicts to involve one row. Removal of that row would then leave $m - 1$ rows that form a GUB set. Thus for c in the range from 1 to $m - 1$, an upper bound on the maximum GUB is $m - 1$. Since one row can conflict with at most $m - 1$ other rows, once $c \geq m$, at least two rows have to be removed to form a GUB set. For $m \leq c \leq [(m-1) + (m-2)]$ it is possible to construct a incident matrix such that all the conflicts are between a pair of rows and the remaining set of rows. Removal of the pair would result in a GUB set of $m - 2$ rows. This constructive argument continues until $c = \frac{(m)(m-1)}{2}$, the maximum number for c. This could occur when each row conflicts with every other row. At that point, the max maximum GUB = min maximum GUB = 1.

A graph of an upper bound on the maximal GUB for a 5 row problem such as the example problem is given below:

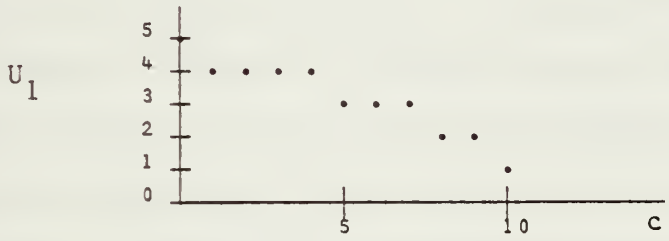

Figure 6

For the example problem, m = 5 and c = 5. From the above graph the upper bound on the maximum GUB for that problem is 3. Since a GUB set containing three rows has already been identified, that set is a maximum set.

In general, for any problem with an m x c incidence matrix, the largest maximum GUB set that can be obtained is:

$$u_1 = \lfloor .5 + \sqrt{.25 + (m)(m-1) - 2c} \rfloor$$

The above bound is *problem-independent* and a *sharp* bound in that matrices with a GUB set the size of the bounding value can be constructed.

With additional information about a specific problem a better bound can be constructed. Since s_i is the number of other rows that conflict with row i, removing row i from the set of rows reduces the number of conflicts, c, by s_i. Let IMAX denote max s_i. Since IMAX is the largest row conflict count, c can be reduced by no more than IMAX with the removal of each row. The minimum number of rows that would have to be removed to reduce the number of row conflicts to 0, is $\lceil c/IMAX$. Therefore, given m, c and IMAX, the bound can be improved to

$$u_2 = \begin{cases} m - \lceil \frac{c}{y} & c \leqslant (m-y)\, y \\ \lfloor .5 + \sqrt{.25 + y\,(2m-y-1) - 2c} & c > (m-y)\,(y) \end{cases}$$

where y = IMAX .

In order to determine IMAX, the entire β vector must be examined.

A third, even better bound can be obtained with additional information on the frequency of the conflict counts from 1 to IMAX. The procedure is the same as above, in that when a row is removed with IMAX conflict count, c decreases by IMAX. However, instead of continuing to decrease c by IMAX, it is decreased by the next largest s_i. This procedure continues until once again, c becomes zero. This bound is named u_3.

Each tighter bound requires more information about the particular problem. However, all the information is readily available since it is generated by the heuristics using the conflict measure.

The bounds developed can be used to objectively evaluate the size of a GUB set found by heuristic methods. In two problems examined, VANN and AIRLP, the number of rows in the GUB set equal an upper bound on the maximal GUB set for the problem. Therefore, for those problems, the heuristic methods are verified to have located maximum GUB sets.

Manual specification of a GUB set from visual inspection can utilize these bounds as an excellent measure of the maximum additional rows to be found. This information is also an aid in deciding whether to subject the problem to additional automatic searching for GUBs.

VIII. EXTENSIONS

The upper bounds developed in this report vary from a problem independent bound, to tighter problem dependent bounds. It is speculated that additional information can be easily extracted from the actual conflict structure of the problems that can be used to tighten the existing bounds even further. In addition, lower bounds can be developed by similar methods.

Another area that warrants further study is the special structure of the incidence matrix representation of the original problem. It is noted that for an incidence matrix, N, the relative weights generated for each row are (except for a constant) identical for both methods studied. This implies that for a matrix N, and the same strategy (i.e., II), the two heuristics would identify the same GUB set.

Finally, research is continuing with automatic location of network row structure. As one illustration of an immediate generalization of the GUB results, a GUB set for a problem can be identified and then another GUB set of an eligible subset of remaining rows can be found. Thus, a bi-partite network row factorization can be achieved (e.g., transportation or assignment rows) . This problem is being further examined by Wright [17].

IX. CONCLUSIONS

The computational benefits of a large GUB set for an LP problem are widely recognized. The need for an algorithm that can extract a maximum GUB from contemporary large scale linear programming models is therefore apparent. This report shows that the identification of a maximum GUB set is a difficult problem, essentially as hard as many other widely known difficult problems.

An alternate approach is the use of an heuristic. This report has examined two promising methods (with two versions of each) with application to a series of real life, large scale models. All versions are robust in their ability to find large GUB sets of rows. However, the two versions (II.9 and II.10) that use the Senju and Toyoda method are consistently the fastest. These two methods are essentially equal in their efficiency and effectiveness. Since version II.9 (which recalculates the gradient after the removal of each row) so selectively removes the rows during the first phase that few if any rows are re-included in the GUB set during the second phase, it suggests the possibility of implementing this version as only a one phase (row deletion) method.

The representation of an infinite number of m row matrices by a finite number of incidence matrices offers a powerful and concise way of examining the GUB problem. Under this representation, both basic heuristic methods investigated assign (within a constant) the same relative selection weights to each row.

Finally, the ability of defining upper bounds on the maximum size of the GUB set gives a new powerful tool in this area. It enables one to evaluate the quality of GUB sets found even in very large problems, for which the algorithmic identification of the maximum GUB set is probably impossible in general. In some cases, verification of a heuristically achieved maximum GUB set is now possible. Further, the bounds developed may be further enhanced in future research, and may be applicable to related problems of equivalent complexity.

Appendix A

This appendix contains the computational results for the fifteen linear, mixed integer and integer models examined. The experiments have been conducted using the FORTRAN H compiler on an IBM 360/67 computer at the W. R. Church computer center of the Naval Postgraduate School. All excution times reported are expressed in actual CPU seconds, accurate to the precision displayed.

For clarity, the following terms are defined:

Eligible rows: The number of rows of the model that were initially eligible for inclusion in a set of GUB rows.

Conflict count: The number of columns of the incidence matrix for the problem, .

Conflict density: The ratio of the conflict count to the maximum conflict count for that problem size. [i.e., (m) (m −1)]

Time to find Elig: The time in CPU seconds to determine the set of eligible rows.

Problem	:	VANN	Description	:	Fleet Dispatch Model		
Rows	:	92	Eligible rows	:	69	IMAX :	0
Columns	:	1324	Conflict count	:	0	U1 :	69
Integer	:	1324	Conflict density	:	0	U2 :	69
Non-zero	:	2648	Time to find Elig	:	.141 sec	U3 :	69

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	69	1324	.237		
II.2	69	1324	.125		
II.9	69	1324	.202	.198	0
II.10	69	1324	.202	.198	0

Problem	:	NETTING	Description	:	Currency Exchange Model		
Rows	:	103	Eligible rows	:	71	IMAX :	5
Columns	:	247	Conflict count	:	46	U1 :	70
Integer	:	103	Conflict density	:	1.85%	U2 :	59
Non-zero	:	494	Time to find Elig	:	.022 sec	U3 :	46

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	36	84	.169		
II.2	36	84	.164		
II.9	36	77	.047	.042	0
II.10	36	72	.042	.037	0

Problem	:	AIRLP	Description	:	Fleet Dispatch Model		
Rows	:	171	Eligible rows	:	170	IMAX :	150
Columns	:	3040	Conflict count	:	2983	U1 :	151
Integer	:	0	Conflict density	:	20.77%	U2 :	150
Non-zero	:	6023	Time to find Elig	:	.076 sec	U3 :	150

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	150	3000	1.16		
II.2	150	3000	.761		
II.9	150	3000	.645	.639	0
II.10	150	3000	.444	.439	0

Problem :	COAL	Description :	Energy Development Model		
Rows :	171	Eligible rows :	170	IMAX :	111
Columns :	3753	Conflict count :	3753	U1 :	146
Integer :	0	Conflict density :	26.13%	U2 :	136
Non-zero :	7506	Time to find Elig :	.106 sec	U3 :	121

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	111	3753	1.38		
II.2	111	3753	1.24		
II.9	111	3753	.920	.912	0
II.10	100	2568	.641	.631	0

Problem :	TRUCK	Description :	Fleet Dispatch Model		
Rows :	239	Eligible rows :	221	IMAX :	171
Columns :	4752	Conflict count :	10438	U1 :	165
Integer :	4752	Conflict density :	42.94%	U2 :	159
Non-zero :	30074	Time to find Elig :	.116 sec	U3 :	144

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	32	1069	6.88		
II.2	30	1099	7.095		
II.9	30	857	5.00	4.95	2
II.10	32	986	1.70	1.58	8

Problem :	CUPS	Description :	Production Scheduling Model		
Rows :	415	Eligible rows :	390	IMAX :	48
Columns :	619	Conflict count :	744	U1 :	388
Integer :	145	Conflict density :	.98%	U2 :	374
Non-zero :	1341	Time to find Elig :	.042 sec	U3 :	294

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	213	494	2.96		
II.2	214	442	3.15		
II.9	214	466	.212	.194	0
II.10	200	394	.384	.132	24

Problem	: FERT	Description	: Production & Distribution Model				
Rows	: 606	Eligible rows	: 605	IMAX	:	580	
Columns	: 9024	Conflict count	: 16455	U1	:	577	
Integer	: 0	Conflict density	: 9.01%	U2	:	576	
Non-zero	: 40484	Time to find Elig	: .257 sec	U3	:	567	

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	559	9024	15.8		
II.2	559	9024	10.5		
II.9	559	9024	6.73	6.71	0
II.10	559	9024	2.52	2.50	0

Problem	: PIES	Description	: Energy Production & Consumption Model				
Rows	: 663	Eligible rows	: 662	IMAX	:	21	
Columns	: 2923	Conflict count	: 4116	U1	:	655	
Integer	: 0	Conflict density	: 1.88%	U2	:	466	
Non-zero	: 13288	Time to find Elig	: .866 sec	U3	:	422	

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	180	1848	10.8		
II.2	169	1693	13.5		
II.9	172	1811	2.82	2.77	1
II.10	177	1761	1.31	.788	28

Problem	: PAD	Description	: Energy Production & Consumption Model				
Rows	: 695	Eligible rows	: 694	IMAX	:	23	
Columns	: 2934	Conflict count	: 4416	U1	:	687	
Integer	: 0	Conflict density	: 1.84%	U2	:	502	
Non-zero	: 13459	Time to find Elig	: .104 sec	U3	:	449	

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	200	1864	13.1		
II.2	189	1771	16.6		
II.9	188	1708	3.34	3.26	2
II.10	189	1275	1.35	.928	21

Problem	:	ELEC	Description	:	Energy Production & Consumption Model		
Rows	:	785	Eligible rows	:	784	IMAX :	22
Columns	:	2800	Conflict count	:	6167	U1 :	776
Integer	:	0	Conflict density	:	2.01%	U2 :	503
Non-zero	:	8462	Time to find Elig	:	.089 sec	U3 :	492

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	309	2461	11.4		
II.2	210	2791	16.1		
II.9	309	2641	1.15	1.12	0
II.10	309	2605	.842	.579	14

Problem	:	GAS	Description	:	Production Scheduling Model		
Rows	:	799	Eligible rows	:	789	IMAX :	608
Columns	:	5536	Conflict count	:	22220	U1 :	760
Integer	:	0	Conflict density	:	7.15%	U2 :	752
Non-zero	:	27474	Time to find Elig	:	.151 sec	U3 :	652

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	583	5102	16.2		
II.2	639	5536	10.4		
II.9	608	5309	3.79	3.77	0
II.10	639	5533	1.47	1.44	1

Problem	:	FOAM	Description	:	Production Scheduling Model		
Rows	:	1017	Eligible rows	:	1006	IMAX :	261
Columns	:	4020	Conflict count	:	8186	U1 :	997
Integer	:	42	Conflict density	:	1.62%	U2 :	974
Non-zero	:	17187	Time to find Elig	:	225	U3 :	934

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	932	4020	23.4		
II.2	932	4020	9.47		
II.9	917	3981	1.73	1.71	0
II.10	917	3981	.902	.879	0

Problem :	LANG	Description :	Equipment & Manpower Scheduling Model		
Rows :	1236	Eligible rows :	1235	IMAX :	184
Columns :	1425	Conflict count :	46424	U1 :	1196
Integer :	0	Conflict density :	6.09%	U2 :	982
Non-zero :	22028	Time to find Elig :	.072 sec	U3 :	973

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	382	1207	46.2		
II.2	338	908	54.2		
II.9	342	923	14.9	14.8	2
II.10	342	922	12.4	1.13	234

Problem :	JCAP	Description :	Production Scheduling Model		
Rows :	2487	Eligible rows :	2446	IMAX :	488
Columns :	3849	Conflict count :	16578	U1 :	2439
Integer :	560	Conflict density :	.55%	U2 :	2412
Non-zero :	9510	Time to find Elig :	.265 sec	U3 :	1812

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	529	2072	104		
II.2	512	2186	153		
II.9	529	2087	2.23	1.87	5
II.10	523	1393	3.98	1.10	59

Problem :	ODAS	Description :	Manpower Planning Model		
Rows :	4648	Eligible rows :	4647	IMAX :	4194
Columns :	4683	Conflict count :	5220	U1 :	4645
Integer :	0	Conflict density :	.05%	U2 :	4645
Non-zero :	30520	Time to find Elig :	.263 sec	U3 :	4024

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
I.2	751	3116	369		
II.2	721	3846	651		
II.9	749	4436	7.12	6.88	0
II.10	751	3020	3.01	2.57	2

LIST OF REFERENCES

[1] Beale, E. M. L., "Advanced Algorithmic Features For General Mathematical Programming Systems," *Integer and Nonlinear Programming*, ed. J. Abadie, North-Holland/American Elsevier, 1970

[2] Brearly, A. L., Mitra, G., and Williams, H. P., "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm," *Mathematical Programming* 8, 54-83, 1975

[3] Brown, G. and Graves, G., "Design and Implementation of a Large Scale (Mixed Integer) Optimization System," paper presented at ORSA/TIMS Las Vegas, Nov. 1975

[4] Dantzig, G. B. and Van Slyke, R. M., "Generalized Upper Bounding Techniques," *Journal of Computer and System Science* 1, 213-226, 1967

[5] Garey, M. R. and Johnson, D. S., *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA., 1979

[6] Graves, G. W. and McBride, R. D., "The Factorization Approach to Large-Scale Linear Programming," *Mathematical Programming* 10, 91-110, 1976

[7] Greenberg, H. J. and Rarick, D. C., "Determining GUB Sets via an Invert Agenda Algorithm," *Mathematical Programming* 7, 240-244, 1974

[8] Hellerman, E. and Rarick, D., "Reinversion with the Preassigned Pivot Procedure," *Mathematical Programming* 1, 195-216, 1971

[9] Hirshfield, D. S., "Generalized Upper Bounding (GUB) Theory, Applications and Performance," paper presented at Share XXXV, Mathematical Programming Project, August 1970

[10] Kaul, R. N. "An Extension of Generalized Upper Bounded Techniques for Linear Programming," (ONR65-27) Operations Research Center, University of California, Berkeley, 1965

[11] Klee, V., "Combinatorial Optimization: What is the State of the ART?" paper presented at SIAM Applied Mathematics Conference, Monterey, CA, February 1978.

[12] Lasdon, L. S., *Optimization Theory for Large Systems*, The MacMillan Company, New York, N. Y. 1970

[13] Lewis, H. R. and Papadimitriou, C. H., "The Efficiency of Algorithms," *Scientific American* 238, No. 1, 96-109, 1978

[14] *MPSIII-Mathematical Programming System User Manual*, Sec. 7, Management Science Systems, Inc., 1973.

[15] McBride, R., "Linear Programming with Linked Lists and Automatic Guberization," Working Paper No. 8175, University of Southern California, School of Business, July 1975.

[16] Senju, S. and Toyoda, Y., "An approach to Linear Programming with 0-1 Variables," *Management Science* 15, B196-B207, 1968

[17] Wright, W., M. S. Thesis, Naval Postgraduate School (in preparation).

[18] Zaks, S. (Private Communication, September 1979)

INITIAL DISTRIBUTION LIST

1	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
4	Associate Professor G.G. Brown, Code 55Bw Department of Operations Research Naval Postgraduate School Monterey, California 93940	2
5	Professor G. H. Bradley, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6	Captain David Samuel Thomen, USMCR 580 Warrenville Road Mansfield Center, Connecticut 06250	1
7	Professor Glenn W. Graves Graduate School of Management University of California Los Angeles, California 90024	1
8	Dr. Neal Glassman Office of Naval Research Code 934 800 N. Quincy Street Arlington, Virginia 22217	1
9	Assistant Professor Shmuel Zaks Department of Computer Science Technion Haifa, Israel	1
10	Associate Professor J. Hartman, Code 55Hh Department of Operations Research Naval Postgraduate School Monterey, California 93940	1